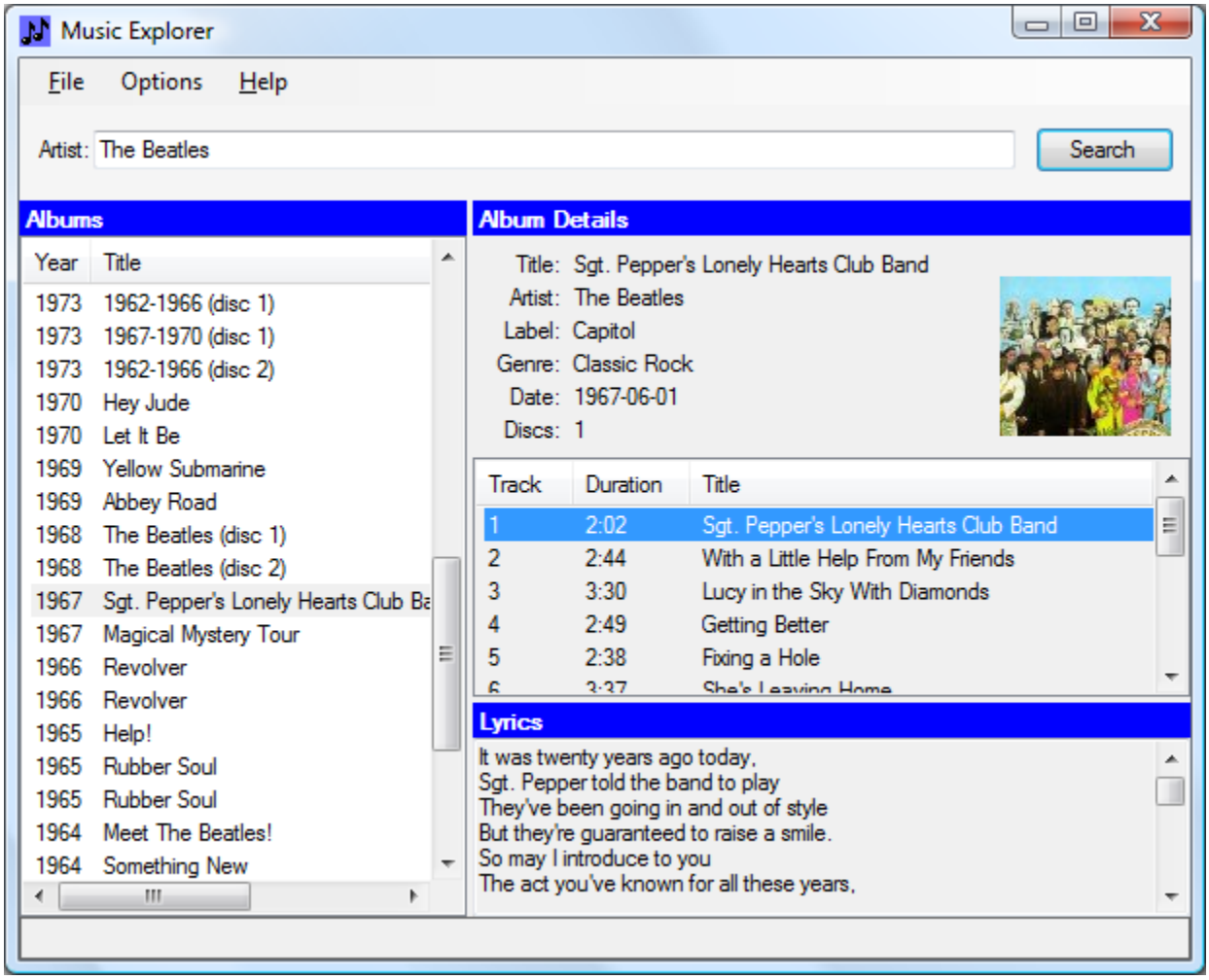
Music Explorer

Ted Faison
2009-04-28



Introduction

Music is a big part of our lives and there is certainly no shortage of it. Hundreds of new music releases occur every day. Never in the history of civilization has so much music been produced, bought, sold and enjoyed. With all that music out there, it can be challenging to keep track of it. I use a number of online services to discover new artists and listen to samples of their music. When I find interesting artists, I like to look up their discographies and download the lyrics of their best songs. I do so with a small Windows program called Music Explorer that I wrote some time ago. Given an artist name, Music Explorer shows you a list of all albums released. For each album, you can get detailed information. For each track on a given album, you can even get the lyrics. The only thing Music Explorer doesn't do is play music, because I didn't want to get embroiled in music download and copyright issues. The following above shows the user interface of Music Explorer.

Music Explorer runs on Windows XP and Vista. Using Music Explorer, you enter the name of an artist or group and click the **Search** button. Within a couple of seconds, you get a list of albums released. By default, the list is sorted by descending year, but can also be sorted by title (by clicking the Title column header). When you select an album, the right pane shows information about the album and its tracks. The album's cover art is also shown. If you select a track, the bottom right pane shows the song's lyrics.

Enter any artist and you can see all the lyrics of his/her songs. Pretty awesome, but Music Explorer is really not that sophisticated. It simply leverages data exposed by a small collection of public web services, showing the data in a (hopefully) intuitive way. Music Explorer is a pretty simple .Net Windows Forms application, written in C#. The application requires no installation procedure. Just copy it into a folder and run it. It doesn't make of any third-party DLLs and will build in Visual Studio 2005 or later.

Web Services Used

Let's look in more detail at the web services called by Music Explorer, since those provide the real added-value of the application. I originally wrote Music Explorer in 2005. Back then, the landscape of music-related web services was relatively small and immature. Amazon had a service called E-Commerce Service (ECS) 3.0 that, given an artist, would return all the albums and tracks. ECS didn't return all the track information I wanted, so I had to call a second web service, MusicBrainz.org, an open source project created by Robert Kaye. For the lyrics, I had to use a third service, which returned lyrics as HTML that I had to scrub before showing to the user. The scrubbing process was somewhat clunky, because occasionally the service would return weird characters in the HTML and mess up the cleansed text.

Fast forward to 2009. Amazon ECS 3.0 is no longer in service. The current version is 4.0 and the name has been changed to *Amazon Web Services* (AWS). The new service has been heavily refactored, to reduce the number of web methods and provide better control over the amount and type of data returned. When ECS 3.0 was turned off, Music Explorer stopped working, so I set it aside with the intention of updating it at some point. That was a while ago, but I finally found some time to update the project.

When I looked at the current AWS 4.0, I discovered that Amazon has a LOT of data that can be returned. To manage retrieval of this data, Amazon resorted to a paging scheme. To give you an idea of how much data I'm talking about, a search for *The Beatles* returned 118 pages of data. It takes way too long to retrieve all those pages, for the purposes of Music Explorer, and most of the data is related to weird limited editions in countries that I'm interested in. The solution was to leverage MusicBrainz.org again. Back in 2005 they supported a data protocol called RDF, which has fallen out of favor. Now MusicBrainz.org exposes a REST XML web service that is pretty easy to use, once you get the hang of it. MusicBrainz also gives pretty impressive performance, giving back a list of albums in a second or two.

Regarding the lyrics web service, I found a newer service called LyricWiki.org that provides exactly what I need. Lyrics are returned as straight text, so I no longer need the HTML

cleansing operation. The LyricWiki database is also pretty extensive. As of this writing, LyricWiki claims to have upwards of 700,000 pages on their site and I'm inclined to believe them. I've poked around with LyricWiki for a while and found that the service indeed seems to be very well stocked.

In a Nutshell

Music Explorer works by calling web services and displaying the information returned. Getting album information for an artist requires the following steps:

1. *Resolving the artist's name*. When the user enters an artist name and then clicks the Search button, the artist name must be resolved to an ID before any other information can be retrieved. The MusicBrainz web service is used.

2. *Getting the artist's albums*. The artist's ID is used to retrieve a list of albums. The MusicBrainz service is used again.

3. *Getting an album's details*. The album IDs (there are two of them) are used to retrieve details, which include things like the Label (e.g. Sony), genre (e.g. Heavy Metal), album cover art and the track listing. The MusicBrainz and Amazon services are used.

4. *Getting a track's lyrics*. The LyricWiki service is used.

The following sequence diagram shows the details of resolving an artist name to an ID.

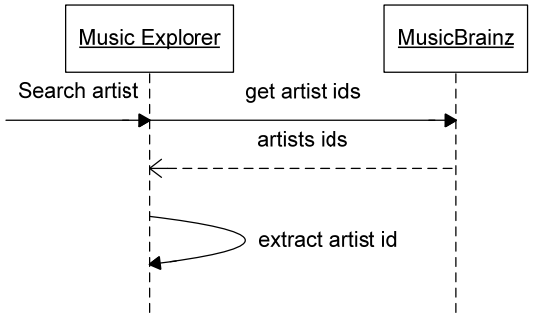


Figure 1 – Resolving an artist name to an ID.

Given an artist name, MusicBrainz returns a list of IDs for artists with similar names. Music Explorer looks for the ID of the artist with the exact name entered. The following sequence diagram shows the details of getting a list of albums for a given artist:

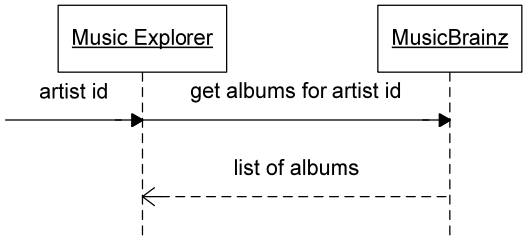


Figure 2 - Getting a list of albums for a given artist.

Each album returned by MusicBrainz includes two IDs: the MusicBrainz ID and the Amazon Standard Identification Number (ASIN). The latter ID is important, because it allows us to access the Amazon Web Service to get any additional information we might need that MusicBrainz doesn't provide.

When the user selects an album, Music Explorer first calls Amazon, using the album ASIN, to get album details such as the genre, the number of discs in the album and the URL of the album cover art. Music Explorer then calls MusicBrainz to get the album's track listing. The following diagram shows the details.

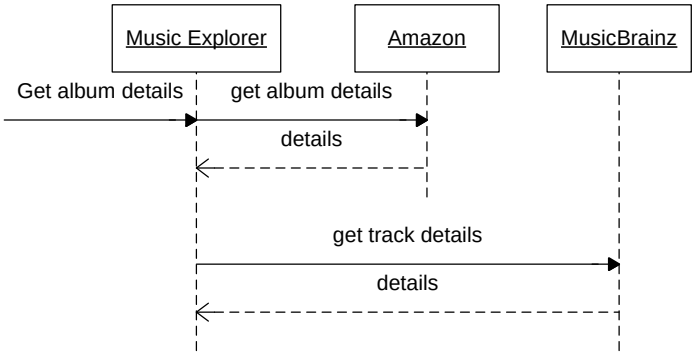


Figure 3 - Getting the details for an album.

When the user selects a track, Music Explorer makes a quick call to LyricWiki to retrieve the lyrics, as described in the next figure. The only parameters LyricWiki needs are the artist name and the song title.

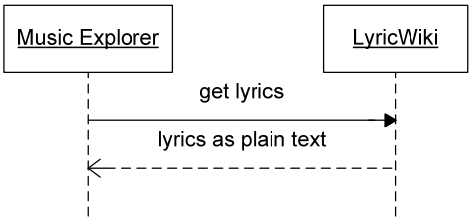


Figure 4 - Getting the lyrics for a song.

LyricWiki returns lyrics in plain text, so you don't have to jump through hoops to clean it up for presentation. Sweet and simple.

The Album and Track Classes

Music Explorer uses an `Album` class to describe music albums. An album represents anything that stores tracks, such as a vinyl record, a cassette or a CD. The class is shown in the following listing:

```
public class Album
{
  public string Id;       // MusicBrainz ID
  public string Asin;     // Amazon Standard Identification Number
  public string Title;    // the title of the album
  public DateTime Date;   // the release date
  public string ImageUrl; // the URL of the cover art picture
  public string Label;    // e.g. Sony, Warner
  public int Discs;       // number of discs in album
  public string Genre;    // e.g. Heavy Metal
  public string Artist;   // name of the artist or group
  public List<Track> Tracks = new List<Track>();
}
```

Listing 1 - The Album class.

The `Tracks` field contains a list of `Track` objects. The `Track` class looks like this:

```
public class Track
{
  public string Id;         // MusicBrainz ID
  public TimeSpan duration; // length of track
  public string Title;      // name of song
}
```

Listing 2 - The Track class.

Resolving an Artist Name to an ID

Let's look at the details. The search process starts by entering an artist name at the top of the Music Explorer screen. The name is used in a call to the MusicBrainz service, with query parameters indicating the name of the artist. The result is XML containing a list of all the artists with names similar to the one entered. For example, to search for artists with names like **Madonna**, Music Explorer would use the following URL:

http://musicbrainz.org/ws/1/artist/?query=Madonna&type=xml

You can enter this URL in a browser, to experiment with the MusicBrainz service. The result is a list of all the matching artists. Each artist is tagged with an **id** attribute. This value is important, because it must be used in subsequent searches for the artist's albums. For the **Madonna** search, MusicBrainz returns the following results:

```
<metadata xmlns="http://musicbrainz.org/ns/mmd-1.0#"
```

```
            xmlns:ext="http://musicbrainz.org/ns/ext-1.0#">
  <artist-list count="11" offset="0">
    <artist id="79239441-bfd5-4981-a70c-55c3f15c1287" type="Person" ext:score="100">
      <name>Madonna</name>
      <sort-name>Madonna</sort-name>
      <life-span begin="1958-08-16" />
    </artist>
    <artist id="71851a56-9a92-422f-9ce2-c16ed0dc6938" ext:score="80">
      <name>Madonna Hip Hop Massaker</name>
      <sort-name>Madonna Hip Hop Massaker</sort-name>
      </artist>
    <artist id="b3d01315-d52a-4f3a-908b-0618315c1ef2" type="Person" ext:score="80">
      <name>Madonna Louise Veronica Ciccone</name>
      <sort-name>Ciccone, Madonna Louise Veronica</sort-name>
      <life-span begin="1958-08-16" />
    </artist>
      ...
    <artist id="6cc0b2d5-e6b5-46c4-82c2-d4b19e721bf8" ext:score="24">
      <name>صـقر جوزف و عرنيطـا مادونـا</name>
      <sort-name>Arnita, Madonna and Joseph Saqr</sort-name>
    </artist>
  </artist-list>
</metadata>
```

Listing 3 - The artist response XML from MusicBrainz.

I omitted most of the <artist> results, for clarity. Note that non-Latin character can appear in the results. Music Explorer uses the following XPATH code to extract the information for the artist with the exact name entered.

```
public static string FindArtistId(string name)
{
  string uri = string.Format(http://musicbrainz.org/ws/1/artist/?query={0}&type=xml",
                             name);
  XPathDocument doc = new XPathDocument(uri);
  XPathNavigator nav = doc.CreateNavigator();
  XmlNamespaceManager nsmgr = new XmlNamespaceManager(nav.NameTable);
  nsmgr.AddNamespace("mb", "http://musicbrainz.org/ns/mmd-1.0#");
  string xpath = string.Format("//mb:artist[mb:name=\"{0}\"]", name);
  XPathNodeIterator ni = nav.Select(xpath, nsmgr);
  if (!ni.MoveNext()) return null;
  XPathNavigator current = ni.Current;
  return current.GetAttribute("id", nsmgr.DefaultNamespace);
}
```

Listing 4 - Extracting the artist's id from the response XML.

The XPATH expression in the highlighted text locates the <artist> element whose name is an exact match to the name entered. XPATH is case sensitive, so searches for “madonna” and “Madonna” will return different results (the former will return none). Once the <artist> element is found, the method in the previous listing returns the value of its “id” attribute, which is what we need.

Finding an Artist's Albums

Once we know the id of an artist, we can call MusicBrainz again to get the list of albums. Using the Madonna example, we would use the following URL:

http://musicbrainz.org/ws/1/artist/79239441-bfd5-4981-a70c-55c3f15c1287?type=xml&inc=sa-Official+release-events

The following listing shows the response:

```
<metadata xmlns="http://musicbrainz.org/ns/mmd-1.0#">
  <artist id="79239441-bfd5-4981-a70c-55c3f15c1287" type="Person">
      <name>Madonna</name>
      <sort-name>Madonna</sort-name>
      <life-span begin="1958-08-16" />
       <release-list>
         <release id="5d9a5a55-e75f-4282-906b-1141f94f1885" type="Album Official">
            <title>Madonna</title>
            <text-representation language="ENG" script="Latn" />
            <asin>B000002KZ3</asin>
            <release-event-list>
              <event date="1983-07-26" country="US" catalog-number="1-23867"
                     barcode="075992386715" format="Vinyl" />
              <event date="1983-07-26" country="US" catalog-number="9 23867-4"
                     barcode="075992386746" format="Cassette" />
              <event date="1983-07-26" country="US" catalog-number="9 23867-2"
                     barcode="075992386722" format="CD" />
              <event date="1983-07-27" country="JP" catalog-number="PKF-5423"
                     format="Cassette" />
            </release-event-list>
          </release>
            ...
            ...
      </release-list>
  </artist>
</metadata>
```

Listing 5 - The album response XML from MusicBrainz.

For each album found (called a *release* in MusicBrainz), you get a list of <release> elements. Each element has a bunch of <event> children, describing the dates, countries and format of each release. The only information Music Explorer needs is the **id** and **asin** of each release. We'll use them to get album details.

Finding the Details for an Album

The most important details we need to know about an album are the track listings and album cover art. Although both are available from Amazon, for some reason Amazon doesn't return the track durations, so Music Explorer uses MusicBrainz to get the track information.

The only parameter we need to supply in the MusicBrainz call is the release ID. The result is XML containing a list of all the tracks. For example, to get the details for Madonna's album titled *Madonna*, we would use the following URL:

http://musicbrainz.org/ws/1/release/5d9a5a55-e75f-4282-906b-1141f94f1885?type=xml&inc=tracks

The result is a list of all the tracks on the album. Each track has a title and duration. For the **Madonna** album search, MusicBrainz returns the following results:

```
<metadata xmlns=http://musicbrainz.org/ns/mmd-1.0#
          xmlns:ext="http://musicbrainz.org/ns/ext-1.0#">
  <release id="5d9a5a55-e75f-4282-906b-1141f94f1885" type="Album Official">
    <title>Madonna</title>
    <text-representation language="ENG" script="Latn" />
    <asin>B000002KZ3</asin>
    <track-list>
      <track id="9a501c4e-c7cb-40f8-ae2d-6cd92807f9a8">
        <title>Lucky Star</title>
        <duration>337106</duration>
      </track>
      <track id="516012bf-ff23-4fb9-ae18-d70f35fa8603">
        <title>Borderline</title>
        <duration>321893</duration>
      </track>
      ...
    </track-list>
  </release>
</metadata>
```

Listing 6 - The album detail response XML from MusicBrainz.

The <duration> values are in milliseconds. The `Track` class stores durations as `TimeSpans`. To get additional album info, Music Explorer calls Amazon. The only parameter needed is the album's ASIN, which we have. The following listing shows the code to get details about the *Madonna* album.

```
ItemLookup search = new ItemLookup();
search.AssociateTag = Properties.Settings.Default.AmazonAwsAccountNumber;
search.AWSAccessKeyId = Properties.Settings.Default.AmazonAwsAccessKeyId;

ItemLookupRequest request = new ItemLookupRequest();
request.ItemId = new string[] { album.Asin };
request.IdType = ItemLookupRequestIdType.ASIN;
request.ResponseGroup = new string[] {"Images","ItemAttributes","BrowseNodes"};
search.Request = new ItemLookupRequest[] { request };

ItemLookupResponse response = amazonWebService.ItemLookup(search);
```

Listing 7 - Calling the Amazon web service to get album details.

As an example, to retrieve the album details for the Madonna album, we would use the ASIN value **B000002KZ3**.

To use the Amazon Web Service, Amazon wants you to have credentials. Not to worry, it won't cost you anything to get your credentials. Just go to http://aws.amazon.com/associates/ and create an account. Amazon will give you an Account Number and an Access Key ID. Once you get these items, enter them in Music Explorer, using the **Options -> Amazon Credentials** dialog box.

NOTE: Music Explorer can also be used without Amazon credentials, but then you won't see some of the album details or the album cover art.

When calling the Amazon web service, you can specify the kinds of things you want in the response. You do so by telling Amazon which `ResponseGroup` items you want. If you don't specify anything, you'll get default items back. In my case, I wanted to get the album cover art, some album details and the album genre information. To do so I specified `Images`, `ItemAttributes` and `BrowseNodes` for the `ResponseGroup`. I'll talk about genres and `BrowseNodes` later. The following listing shows how Music Explorer processes the Amazon response, extracting the details needed.

```
AmazonAws.Items items = response.Items[0];
if (items.Item == null) return;
if (items.Item.Length == 0) return;
AmazonAws.Item item = items.Item[0];
if (item == null) return;

if (item.MediumImage == null)
  album.ImageUrl = null;
else
  album.ImageUrl = item.MediumImage.URL;

if (item.ItemAttributes != null)
  album.Label = item.ItemAttributes.Label;

if (item.ItemAttributes.NumberOfDiscs != null)
  album.Discs = int.Parse(item.ItemAttributes.NumberOfDiscs);

album.Genre = GetGenre(item);
```

Listing 8 - Processing the album details returned by Amazon.

A couple of words are needed to explain how the album's genre is obtained. Amazon returns genre information (and other data) in a `BrowseNode` list. An album can have several genres. Theoretically each track could be a different genre, or be associated with multiple genres. For the purposes of Music Explorer, I am only interested in the main genre, the one that summarizes the genre for the album as a whole. After some experimentation, I found that this info can be obtained by looking at the ancestor node of the *general* genre. The following listing shows the details.

```
static string GetGenre(AmazonAws.Item item)
{
  if (item.BrowseNodes == null) return null;

  string genre = string.Empty;
```

```
  foreach (BrowseNode node in item.BrowseNodes.BrowseNode)
  {
    if (node.Name.ToLower() == "general")
      if (node.Ancestors != null)
        return node.Ancestors[0].Name;
  }
  return "<unspecified>";
}
```

Listing 9 - Determining the main genre of an album.

Getting the Cover Art for an Album

The Amazon Web Service returns three images for the cover art - for small, medium and large sizes. Music Explorer only uses the medium-sized image. The Amazon service only returns the URLs of images, not the images themselves. The image files are stored on Amazon servers, so once we have the image URL it is a simple matter to retrieve the actual image. The next listing is for the `AlbumDetails.ShowCoverArt()` method. The class uses a field named `album` to track the current album.

```
void ShowCoverArt()
{
  if (album.ImageUrl == null)
  {
    pictureBoxAlbumCover.Image = notAvailable;
    return;
  }

  WebClient client = new WebClient();
  client.Headers["User-Agent"] = "Mozilla/4.0";

  byte[] bytes = client.DownloadData(album.ImageUrl.ToString());
  MemoryStream stream = new MemoryStream(bytes);
  pictureBoxAlbumCover.Image = System.Drawing.Image.FromStream(stream);
}
```

Listing 10 – Retrieving an album's cover art image from Amazon.

If no URL is available, Music Explorer uses a default bitmap called `notAvailable`, which is stored in the project's resources. The image contains the text *Album cover not available*. Without a default image, the cover art `PictureBox` would simply appear empty.

User Interface Structure

Music Explorer has a simple but effective appearance, based on a number of resizable panels hosted by a Windows Form named `FormMain`. The following figure shows the panel composition.

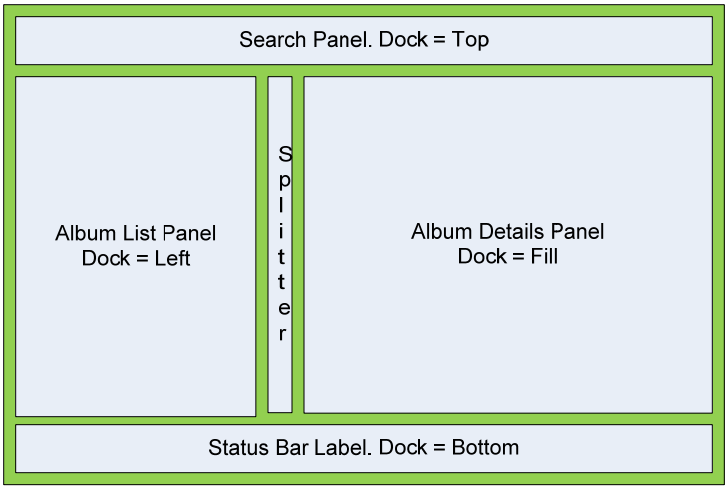


Figure 5 - The layout of the Music Explorer main form.

The Album List panel on the left size contains a ListView populated with the artist's albums. ListView items have a `Tag` property, which Music Explorer uses to store `Album` objects. When the user selects an album, the SelectedIndexChanged event fires. The handler retrieves the `Album` object from the ListView `Tag` and passes it to the `AlbumDetails` user control, which then shows all the album information, tracks and lyrics.

The main window has a vertical splitter to resize the album list. The Album Details panel on the right side hosts the `AlbumDetails` user control, which in turn hosts its own resizable panels, as shown in the next figure.

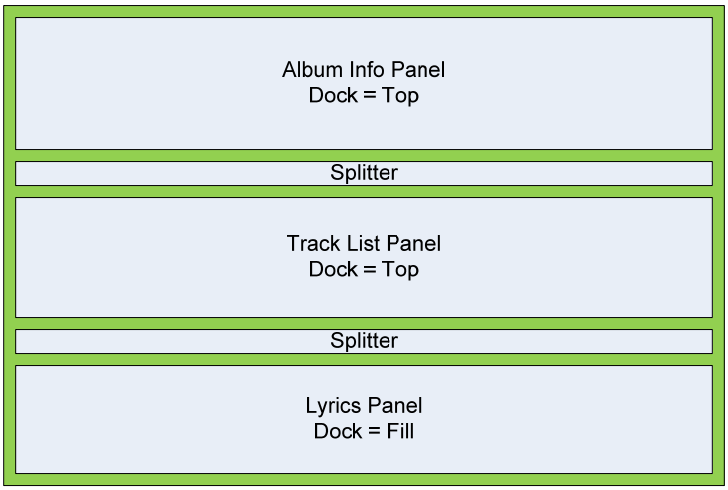


Figure 6 - The layout of the AlbumDetails user control.

The `AlbumDetails` user controls has 2 splitters, to adjust the height of the top and middle panels. The panel at the bottom fills in the available space left.

Implementation Notes

Not all the features of Music Explorer are obvious by looking at its user interface. The following list provides some additional information that may be of interest.

- *ListView Autosizing*. Music Explorer uses two ListViews: one for the Albums list and one for the Tracks list. I wanted the Listviews to autosize their rightmost column, to fill the available space. Autosizing is achieved through a somewhat undocumented feature of the ListView `ColumnHeader` class: all you have to do is set the width of the rightmost column to -2.

- *User Settings*. Music Explorer saves the following settings: your two Amazon credentials (if supplied) and the name of the artist searched. Settings are stored with user scope, meaning that if two different Windows users both use Music Explorer, they each have their own settings. The settings are not stored in an app.config file; they are stored in a user-specific folder. Under Vista, the folder is

  ```
  c:\Users\<user name>\AppData\Local\MusicExplorer
  ```

  The settings are stored in a deep subdirectory of this folder, in a file named `user.config`.

- *Exception Handling*. Music Explorer has rather limited exception handling, to keep the code as clean as possible. The purpose of Music Explorer was really not to create a production-level product. It basically only handles exceptions that occur during web service calls, since these are the most likely points of failure.

- *Album List Contents*. The album list doesn't show all the albums returned by MusicBrainz. After playing with the MusicBrainz service a bit, I noticed that some albums don't have an ASIN value. This problem doesn't prevent Music Explorer from listing the album, but from experimentation it looks like albums without an ASIN are fairly weird. I suspect that such albums are related to special limited releases, such as promotions or country-specific releases. Also, without an ASIN, Music Explorer can't retrieve cover art for an album. For all these reasons, I opted to ignore ASIN-less albums in the album list.

- *Lyrics Text Handling*. The Lyrics panel is clipboard-aware. To copy lyrics to the clipboard, select the lyrics text. Either hit Control-V or use a right-click to bring up the context menu, on which you'll use the Copy menu.

- *No Multi-Threading*. My first implementation of Music Explorer, back in 2005, made use of asynchronous calls to the web services. At the time the services were fairly slow, particularly the Amazon ECS 3 service. Having switched to MusicBrainz for the album list, I found the service to be very fast, returning results typically in a second or two. Getting track information and lyrics takes even less, so I decided to simplify Music Explorer and remove the asynchronous calls. Although the web service calls block the UI thread momentarily, the pause is so short to be of no consequence to users, unless the web service call hangs for some reason.

Conclusion

I have seen lots of Internet posts and blogs regarding retrieval of info from MusicBrainz and Amazon AWS, which tells me that there's quite of bit of interest in leveraging publicly available web services to get music-related information. There are several other web services out there, so you might want to try services other than MusicBrainz, Amazon and LyricWiki. Music Explorer is designed in such a way to make it relatively easy to switch to other services, but Amazon was the only service I found that supports album cover art. Hopefully you'll find Music Explorer helpful in your music quest. I know I have. If I get some time in the future, perhaps I'll port the code to other languages, like VB.Net or Ruby.